This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full "License and Conditions of Use" notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

Annex C. General Reference Architecture in Python for Institutional Integration of the Deterministic Quantum Hash Framework

Purpose

This annex proposes a general institutional reference architecture in Python for implementing the deterministic quantum hash framework, with emphasis on interoperability, provenance, governance, and explainability.[file:202][web:255][web:262][web:261] The architecture is designed so that universities, laboratories, hospitals, public agencies, or research consortia can integrate the model into existing scientific data systems without replacing their current workflows.[web:239][web:262]

C.1 Architectural Rationale

The deterministic quantum hash framework is not only a formula but a full integration object: it requires ingestion of state descriptors, normalization, operator evaluation, entropy calculation, time and causal encoding, metadata governance, canonical serialization, and final cryptographic hashing.[file:202] For institutional adoption, these responsibilities should be separated into modular services so that each institution can adapt source systems and domain rules while preserving a stable core implementation.[file:202][web:255][web:262]

A Python reference architecture is appropriate because Python already dominates scientific computing, workflow orchestration, metadata tooling, AI integration, and knowledge-graph interfacing in many research environments.[web:255][web:261][web:265][web:267]

C.2 General Architectural Principles

The proposed architecture follows six principles:[file:202][web:255][web:262]

- **Modularity:** each model responsibility is isolated in a replaceable component.
- **FAIR compatibility:** metadata and outputs should remain findable, accessible, interoperable, and reusable.[web:262][web:265]

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full "License and Conditions of Use" notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

- **PROV alignment:** provenance should be captured in a queryable lineage model compatible with established provenance approaches.[web:255][web:261]
- **Schema governance:** field names, units, serialization, and versioning must be controlled institutionally.[file:202]
- **Human-auditable AI:** AI may assist integration, but every transformation affecting the final deterministic hash should be inspectable.[web:253][web:260]
- **Explainability-by-design:** each computed hash should be decomposable into the components that generated it.[file:202]

C.3 Reference Architecture Overview

The institutional reference architecture contains eight layers:[file:202][web:262][web:261]

1. **Source connectors layer** for databases, files, APIs, instruments, and workflow systems.
2. **Standardization layer** for schema enforcement, unit conversion, metadata harmonization, and missing-data diagnostics.
3. **Model core layer** for state normalization, effective operator evaluation, entropy computation, energy scaling, temporal indexing, and causal coding.[file:202]
4. **Serialization and cryptographic layer** for canonical input construction and final hash generation.[file:202]
5. **Provenance layer** for trace capture, execution history, and lineage storage.[web:255][web:261][web:262]
6. **Explainability layer** for component attribution, indicator-to-model mapping, and human-readable reports.[file:202]
7. **AI support layer** for metadata linking, anomaly detection, mapping assistance, and graph enrichment.[web:253][web:260]
8. **Institutional governance layer** for roles, approvals, configuration policies, and model version control.[file:202]

C.4 Python Package Structure

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

A practical Python implementation can be organized as follows:

```
quantum_hash_platform/
├── app/
│   ├── api/
│   │   ├── routes_hash.py
│   │   ├── routes_explain.py
│   │   └── routes_admin.py
│   ├── core/
│   │   ├── models.py
│   │   ├── validators.py
│   │   ├── state_builder.py
│   │   ├── operator_engine.py
│   │   ├── entropy_engine.py
│   │   ├── scaling_engine.py
│   │   ├── causal_engine.py
│   │   ├── serializer.py
│   │   ├── hash_engine.py
│   │   └── explainability.py
│   ├── connectors/
│   │   ├── nist_connector.py
│   │   ├── repository_connector.py
│   │   ├── workflow_connector.py
│   │   └── local_lab_connector.py
│   ├── provenance/
│   │   ├── prov_recorder.py
│   │   ├── lineage_store.py
│   │   └── graph_export.py
│   ├── ai/
│   │   ├── metadata_harmonizer.py
│   │   ├── mapping_assistant.py
│   │   ├── anomaly_detector.py
│   │   └── report_generator.py
│   ├── governance/
│   │   ├── config_registry.py
│   │   ├── schema_registry.py
│   │   └── approval_rules.py
│   └── main.py
├── tests/
├── configs/
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

```
├── docs/
└── pyproject.toml
```

This structure keeps the deterministic core stable while allowing institutions to swap connectors, AI assistants, and governance rules according to their infrastructure.[file:202][web:262]

C.5 Core Data Model in Python

The following code illustrates a general data model for institutional integration:

```python
from dataclasses import dataclass, field
from typing import Any, Dict, List, Optional
from datetime import datetime

@dataclass
class StateDescriptor:
    system_id: str
    system_class: str
    timestamp: datetime
    rho: Optional[Any] = None
    probability_vector: Optional[List[float]] = None
    operator_definition: Optional[Dict[str, Any]] = None
    entropy_method: str = "von_neumann"
    energy_value: Optional[float] = None
    energy_unit: str = "J"
    causal_code: Optional[str] = None
    metadata: Dict[str, Any] = field(default_factory=dict)

@dataclass
class NormalizationConfig:
    schema_version: str
    s_max: float
    s_ref: float
    kappa: float
    gamma: float
    delta_t_seconds: float
    serialization_format: str = "canonical_json"
    hash_algorithm: str = "sha3_256"
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

```
@dataclass
class HashResult:
    system_id: str
    re_hq: float
    im_hq: float
    hash_value: str
    normalized_entropy: float
    energy_log10: float
    time_index: int
    causal_code: str
    provenance_id: str
    explanation: Dict[str, Any]
```

This data model provides a clear institutional contract between source systems, the model core, and downstream provenance services.[file:202]

C.6 Core Processing Pipeline in Python

The following reference code illustrates the central computation flow:

```
import json
import math
import hashlib
from typing import Dict

class QuantumHashService:
    def __init__(self, config: NormalizationConfig):
        self.config = config

    def build_state(self, state: StateDescriptor):
        if state.rho is not None:
            return self._normalize_density_matrix(state.rho)
        if state.probability_vector is not None:
            total = sum(max(x, 0.0) for x in state.probability_vector)
            return [max(x, 0.0) / total for x in state.probability_vector]
        raise ValueError("A valid state representation is required")

    def compute_entropy(self, normalized_state) -> float:
        if isinstance(normalized_state, list):
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

```
            values = [x for x in normalized_state if x > 0]
            return -sum(x * math.log(x) for x in values)
        eigvals = [x for x in self._eigenvalues(normalized_state) if x > 0]
        return -sum(x * math.log(x) for x in eigvals)

    def normalize_entropy(self, entropy: float) -> float:
        if self.config.s_max <= 0:
            raise ValueError("s_max must be positive")
        value = entropy / self.config.s_max
        return min(1.0, max(0.0, value))

    def compute_energy_log10(self, energy_value: float) -> float:
        if energy_value is None or energy_value <= 0:
            raise ValueError("energy_value must be positive")
        return math.log10(energy_value)

    def compute_time_index(self, ts: datetime, t0: datetime) -> int:
        dt = (ts - t0).total_seconds()
        return int(round(dt / self.config.delta_t_seconds))

    def compute_correction(self, s_norm: float) -> float:
        return self.config.kappa * (s_norm - self.config.s_ref)

    def continuous_descriptor(self, energy_log10: float, s_norm: float):
        re_hq = energy_log10 + self.compute_correction(s_norm) / 100.0
        im_hq = self.config.gamma * s_norm
        return re_hq, im_hq

    def serialize_payload(self, payload: Dict) -> str:
        if self.config.serialization_format != "canonical_json":
            raise ValueError("Unsupported serialization format")
        return json.dumps(payload, sort_keys=True, separators=(",", ":"), ensure_ascii=False)

    def hash_payload(self, serialized: str) -> str:
        if self.config.hash_algorithm != "sha3_256":
            raise ValueError("Unsupported hash algorithm")
        return hashlib.sha3_256(serialized.encode("utf-8")).hexdigest()

    def process(self, state: StateDescriptor, t0: datetime, provenance_id: str) -> HashResult:
        normalized_state = self.build_state(state)
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

```
    entropy = self.compute_entropy(normalized_state)
    s_norm = self.normalize_entropy(entropy)
    energy_log10 = self.compute_energy_log10(state.energy_value)
    time_index = self.compute_time_index(state.timestamp, t0)
    re_hq, im_hq = self.continuous_descriptor(energy_log10, s_norm)

    payload = {
        "system_id": state.system_id,
        "system_class": state.system_class,
        "re_hq": re_hq,
        "im_hq": im_hq,
        "time_index": time_index,
        "causal_code": state.causal_code,
        "metadata": state.metadata,
        "schema_version": self.config.schema_version,
        "provenance_id": provenance_id,
    }

    serialized = self.serialize_payload(payload)
    hash_value = self.hash_payload(serialized)

    explanation = {
        "energy_log10": energy_log10,
        "normalized_entropy": s_norm,
        "re_hq": re_hq,
        "im_hq": im_hq,
        "time_index": time_index,
        "causal_code": state.causal_code,
        "serialization_format": self.config.serialization_format,
        "hash_algorithm": self.config.hash_algorithm,
    }

    return HashResult(
        system_id=state.system_id,
        re_hq=re_hq,
        im_hq=im_hq,
        hash_value=hash_value,
        normalized_entropy=s_norm,
        energy_log10=energy_log10,
        time_index=time_index,
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

```
        causal_code=state.causal_code or "UNSPECIFIED",
        provenance_id=provenance_id,
        explanation=explanation,
    )

def _normalize_density_matrix(self, rho):
    return rho

def _eigenvalues(self, rho):
    return []
```

This code is intentionally general: institutions can replace the state normalization and eigenvalue routines with NumPy, SciPy, JAX, PyTorch, or domain-specific quantum libraries while preserving the reference architecture.[file:202]

C.7 API Layer for Institutional Use

A minimal institutional deployment can expose the framework through a Python API based on FastAPI or a similar service layer.[web:262] The key endpoints are:

- `POST /hash/compute` for new deterministic hash generation.
- `POST /hash/explain` for explanatory decomposition of a computed hash.
- `POST /hash/validate` for schema and source validation.
- `GET /provenance/{id}` for lineage retrieval.
- `GET /config/version/{version}` for reproducibility of institutional parameter sets.[file:202]

This service-oriented structure allows laboratories or institutions to integrate the framework with existing data platforms, internal portals, electronic laboratory notebooks, or workflow orchestration systems.[web:255][web:262]

C.8 Provenance and Governance Layer

Institutional integration is strongest when every computed hash is accompanied by formal provenance and governance records.[web:255][web:261][web:262] The following information should be stored with each hash event:[file:202]

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full "License and Conditions of Use" notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

- source dataset identifiers,
- normalization configuration version,
- operator-definition version,
- entropy-estimation method,
- serializer version,
- hash algorithm version,
- user or service account,
- approval or validation status.

This aligns the framework with provenance-aware scientific architecture and makes it compatible with queryable institutional traceability systems.[web:255][web:261]

C.9 AI Support Modules for Institutions

AI can be incorporated safely as an auxiliary institutional layer rather than as a replacement for the deterministic core.[web:253][web:260] In practice, Python AI modules can help with:

- metadata harmonization across repositories,
- candidate source linking,
- anomaly detection in state descriptors,
- automated generation of explanation reports,
- ontology or knowledge-graph enrichment of provenance records.[web:253][web:258][web:264][web:267]

This is important because institutions often operate heterogeneous infrastructures where different departments encode similar scientific objects in incompatible ways.[web:262][web:265] AI can reduce this friction while the deterministic model core continues to produce verifiable outputs.[file:202]

C.10 Explanatory Power of the Model

C.10.1 Why the Model Is Explanatorily Strong

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

The explanatory power of the framework comes from the fact that the final identifier is not an opaque checksum detached from scientific meaning.[file:202] Instead, the hash is generated from a structured intermediate representation in which each component has interpretable content: state structure, effective system response, disorder or informational complexity, macroscopic scale, temporal location, causal admissibility, and contextual provenance.[file:202]

This gives the model explanatory strength on three levels:[file:202]

- **Physical level:** the framework preserves interpretable scaling through E and domain-defined effective observables.
- **Informational level:** it expresses configurational disorder through entropy and normalized entropy.
- **Operational level:** it records when, under which causal and metadata conditions, the system identity was assigned.[file:202]

C.10.2 Explainability Output Example

The architecture should generate an explanation object for every computed hash, for example:

```
{
  "system_id": "SYS-001",
  "hash_value": "ab4f...",
  "components": {
    "state_representation": "density_matrix",
    "energy_log10": 41.73,
    "normalized_entropy": 0.42,
    "re_hq": 41.7312,
    "im_hq": 0.42,
    "time_index": 1024,
    "causal_code": "VALID",
    "operator_version": "op_v2.1"
  },
  "source_trace": {
    "energy_source": "NIST/CODATA + institutional mass record",
    "state_source": "simulation output",
    "metadata_source": "registry-v5"
  },
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

```
  "decision_notes": [
    "entropy normalized with class-specific s_max",
    "canonical JSON serialization applied",
    "sha3_256 selected by policy"
  ]
}
```

Such an explanation object increases institutional trust because the model output can be audited, reproduced, and interpreted rather than merely consumed as a static identifier.[file:202][web:253]

C.11 Institutional Deployment Modes

Three deployment modes are recommended:[file:202]

- **Local laboratory mode:** a single Python service connected to one repository and one provenance database.
- **Institutional platform mode:** a central service integrated with workflow systems, metadata registries, and approval rules.
- **Consortium mode:** federated Python services sharing schema versions, operator registries, and provenance standards across institutions.[web:262][web:265]

This flexibility allows phased adoption while preserving a single reference logic.[file:202]

C.12 Conclusions

A Python-based institutional reference architecture for the deterministic quantum hash framework is fully feasible with current technology and aligns naturally with provenance-aware scientific systems, FAIR-oriented metadata pipelines, and explainable AI-assisted infrastructure.[web:255][web:261][web:262][web:265] The most robust design is modular, provenance-centered, schema-governed, and auditable, with AI serving as an integration assistant rather than as a substitute for the deterministic core.[file:202][web:253]

The explanatory power of the model is one of its strongest institutional advantages because it transforms a hash from a purely syntactic identifier into a structured scientific identity object linked to physical scale, informational complexity, temporal context, causal constraints, and provenance conditions.[file:202] For institutions seeking traceable and interoperable scientific identity across

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full "License and Conditions of Use" notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

workflows, simulations, datasets, and biological pipelines, this architecture provides a realistic and extensible implementation path.[file:202]

LICENSE AND CONDITIONS OF USE.
The content is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) license:
https://creativecommons.org/licenses/by-nc-sa/4.0/
By using this content and model, you agree to:
Provide appropriate credit, include a link to the above license, and indicate if changes were made.
Not use the material or any derivative works for commercial purposes.
Distribute any adaptations or derivative works only under the same CC BY-NC-SA 4.0 license.
Not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
Data logging and trade secrets policy.
The design, implementation, and deployment of this model, as well as any derivatives, do not authorize the collection, storage, or exploitation of IP addresses, unique device identifiers, or other technical traceability data for commercial profiling, targeted advertising, or any other economic exploitation.
Any system implementing this model must limit the logging and retention of IP addresses and other identifiable data strictly to what is necessary to:
(a) maintain technical security and service integrity; and
(b) comply with applicable legal obligations (for example, any minimum log retention required by law).
It is prohibited to use this model or its derivatives to:
capture, store, or exploit trade secrets, proprietary know-how, or confidential information of users or third parties for any purpose other than providing the technical service described in this documentation; or train or improve closed, commercial systems using data that contains confidential information of third parties without their prior, explicit, written consent.

---+EOD+---